Shell expansion: proper use and advanced forms

Tyson Whitehead

February 3, 2016

Read evaluate print

The shell is essentially a read-evaluate-print loop (REPL) for manipulating a state-full environment and starting programs.

1. a line of text is read,
2. the results is evaluated,
3. any relevant output is printed, and
4. the process loops (repeats).

Command evaluation

The basic building block of a shell command is the simple command

[assignment . . .] [word . . .] [redirection . . .]

which is evaluated as follows:

1. Words that are not variables assignments or redirections are expanded.

   - first word becomes command
   - remaining become arguments

2. Redirections are performed.

Command evaluation (cont.)

3. Variables are expanded and assigned.

- doesn't include brace and process substitution
- assignment is for command if there is a command and shell otherwise

4. Alias expansion is applied if the command was not quoted.

Command evaluation (cont.)

5. The identified function, builtin, or external program is executed with the arguments.

- external programs are executed in a separate inherited environment

Expansion

1. brace expansion
2. tilde expansion, parameter and variable expansion, command substitution, arithmetic expansion
3. word splitting
4. pathname expansion

Expansion (cont)

Brace expansion

Row-major (last fastest varying) prefix and suffixed pattern expansion.

pre{str1,. . . }suf

pre{x..y[..inc]}suf

Expansion (cont)

Tild expansion

Substitute directory.

~	home directory of logged in user
~user	home directory of specified user
~n	n'th directory on dir stack

Expansion (cont)

Parameter and variable expansion

Parameters are entries that store values (integers, names, and special characters). Variables are named parameters.

- ! prefix introduces level of indirection
- quotation stop word splitting
- possible to create reference variables (`declare -n`)

Expansion - parameter and variable (cont)

*	all positional parameters (single word when quoted)
@	all positional parameters (multiple words when quoted)
n	nth positional parameter
#	number of position parameters
?	exit status of most recent foreground pipeline
-	current option flags
$	process ID of shell
!	process ID of most recent background
_	last argument to previous command

Expansion - parameter and variable (cont)

name	variable
name[*]	array all entries (single word when quoted)
name[@]	array all entries (multiple words when quoted)
name[subscript]	array single entry

Expansion - parameter and variable (cont)

Value

Substitute value.

$parameter

${parameter}

Expansion - parameter and variable (cont)

Default

Substitute error, default, alternative, assignment.

- : acts on null as well as unset

```
${parameter?word} ${parameter:?word}

${parameter-word} ${parameter:-word}

${parameter+word} ${parameter:+word}

${parameter=word} ${parameter:=word}
```

Expansion - parameter and variable (cont)

Subscripts

Substitute all keys, matching keys.

- quoted * form expands to single argument
- quoted @ form expands to multiple argument

```
${!name[@]} ${!name[*]}
```

```
${!prefix*} ${!prefix@}
```

Expansion - parameter and variable (cont)

String/array subset

Substitute length, subset.

${#parameter}

${parameter:offset} ${parameter:offset:length}

Expansion - parameter and variable (cont)

Prefix/suffix removal, search and replace.

Substitute with removal, search and replace.

- double variant is longest/all matching

${parameter#word} ${parameter##word}

${parameter%word} ${parameter%%word}

${parameter/pattern/string} ${parameter//pattern/string}

Expansion - parameter and variable (cont)

Up/down-case

Matched character case modification.

- pattern applied to each character
- double variant is all matching

${parameter^pattern} ${parameter^^pattern}

${parameter,pattern} ${parameter,,pattern}

Expansion (cont)

Command substitution

Execute command and substitute output.

- $(<file) is alternative to $(cat file)

$(command) ‘command‘

Expansion (cont)

Arithmetic expansion

Evaluate expression.

$((expression))

name	variable
n	number
0n	octal number
0xn 0Xn	hex number
base#n	base-n number

Expansion - arithmetic expansion (cont)

id++ id–	post-increment/decrement
++id –id	pre-increment/decrement
- +	unary sign
! ~	logical/bitwise negation
**	exponential
* / %	multiplication, division, remainder
+ -	addition subtraction
<< >>	left/right binary shift

Expansion - arithmetic expansion (cont)

`<= >= < >`	comparison
`== !=`	equality inequality
`&`	bitwise AND
`^`	bitwise XOR
`|`	bitwise OR
`&&`	logical AND
`||`	logical OR

Expansion - arithmetic expansion (cont)

expr?expr:expr	conditional
= *= /= %= += -= <<= >>= &= ^= |=	assignment
expr1,expr2	sequence

Word splitting

Unquoted expansions are split into words delineated by IFS characters. Unsetting IFS turns this off.

- lack of quoting means lots of scripts don't handle spaces

Word splitting (cont)

Quoting

\	preserves literal meaning of single character
'. . .'	preserves literal meaning of all characters
". . ."	preserves literal meaning of all characters but $, ‘, \, and (possibly) ! which retain their normal meaning
$'string'	ANSI C backslash characters expanded
$"string"	translate according to current locale

Pathname expansion

Row-major (last fastest varying) matching directory entires.

- no match leaves pattern by default

*	match any string
?	match any character
[. . .]	match any enclosed character
[ˆ...]	invert sense of match

Pathname expansion (cont)

Characters

- collation range is only ASCII if LANG=C

c	match single character
a-f	match any character in collation range
[:class:]	range defined by POSIX class
[=c=]	range equivalent collation weight of c
[.symbol.]	match collating symbol